MAT495 Final Report: The PCP Theorem

Andrew Feng

April 2022

The PCP theorem in is a cornerstone result in complexity theory that says two things: first, **NP** has proofs that can be checked probabilistically using "few" random bits and constant queries (as low as 3); second, some **NP**-hard optimization problems cannot have "good" polynomial time approximation algorithms, unless $\mathbf{P} = \mathbf{NP}$.

In this short report, I will explain precisely what the above means and why they are equivalent. Finally, I will prove a weaker version of it. The material in this report mainly came from Arora and Barak's book on computational complexity.

0.1 Probabilistically Checkable Proof

The first point of view is called *probabilistic checkable proof.*

Definition. Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say L has a (r, q)-verifier if there is a polynomial-time probabilistic Turing machine V such that

- Efficiency: On input $x \in \{\texttt{0},\texttt{1}\}^n$ and given access to a proof string $\pi \in \{\texttt{0},\texttt{1}\}^*$ of length at most $q(n)2^{r(n)}$, V uses at most $r(n)$ random bits and $q(n)$ nonadaptive queries to bits of π. Then, V returns a bit in time polynomial in n. Here, nonadaptive means queries do not depend on previous queries.
- Completeness: If $x \in L$, then there is some π that makes V always output 1.
- Soundness: If $x \notin L$, then, for every π, V outputs 0 with probability at least 1/2.

A language L is in $\mathbf{PCP}(r, q)$ if it has a (cr, dq)-verifier for some positive constants c, d.

The above definition roughly captures what it means for a language to have membership proofs that can be checked probabilistically with high confidence. Notice the constant 1/2 is arbitrary in the above definition since repeating the algorithm can reduce the error rate.

In this view, the PCP theorem says

Theorem (PCP Theorem)**.** $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

One inclusion of this theorem is in fact trivial—$\mathbf{PCP}(\log n, 1) \subseteq \mathbf{NP}$. Suppose L has a $(c \log n, d)$-verifier V for some positive constants c, d. Given $x \in \{\texttt{0},\texttt{1}\}^n$, do the following. Nondeterministically write down a proof π of length dn^c. Loop through all binary strings r of length at most $c \log n$ and run V on x, π, r. Even though V has random access to r, it can still be simulated in polynomial time since π is short. If V accepts for all r, accept; otherwise, reject. This is a nondeterministic polynomial time algorithm for deciding L, so $L \in \mathbf{NP}$.

0.2 Hardness of Approximation

The second point of view is hardness of approximation. It tells us unless $\mathbf{P} = \mathbf{NP}$, there are **NP**-hard optimization problems that cannot be approximated in polynomial time to some approximation ratio. Let us make this precise.

Consider the optimization version of `3Sat`, `Max-3Sat`. The goal of this problem is to find an assignment of variables in formula ϕ that maximizes the number of clauses evaluating to true. If a formula ϕ has m clauses and the best such assignment satisfies k of them, then we define the value $\texttt{val}(\phi) = k/m$.

Then, an algorithm is said to be a ρ-approximation algorithm for `Max-3Sat` if for every `3Cnf` formula ϕ, the algorithm returns an assignment satisfying at least $\rho \cdot \texttt{val}(\phi)$ clauses.

Theorem (PCP Theorem as hardness of approximation)**.** *There exists a constant $\rho < 1$ such that for any $L \in \mathbf{NP}$, there is a polynomial time computable function $f : \{\texttt{0},\texttt{1}\}^* \to \{\texttt{0},\texttt{1}\}^*$ satisfying*

- *if* $x \in L$*, then* $\mathtt{val}(f(x)) = 1$*;*
- *if* $x \notin L$*, then* $\mathtt{val}(f(x)) < \rho$*.*

The hardness of approximation view is highlighted by the following corollary.

Corollary. *There is a constant* $\rho < 1$ *such that the existence of a* ρ*-approximation algorithm for* `Max-3Sat` *implies* $\mathbf{P} = \mathbf{NP}$.

Proof. Let ρ be the constant given by the PCP theorem and suppose there is a ρ-approximation algorithm M for `Max-3Sat`. Let $L \in \mathbf{NP}$ and let f be the poly-time computable function given by the PCP theorem for L. An algorithm for deciding L is: on input x, return 1 if and only if $M(f(x))$ satisfies at least a ρ fraction of clauses of $f(x)$. If $x \in L$, then $f(x)$ has $\mathtt{val}(f(x)) = 1$, so $M(f(x))$ satisfies at least a $\rho \cdot \mathtt{val}(f(x)) = \rho$ fraction of clauses of $f(x)$. If $x \notin L$, then $f(x)$ has $\mathtt{val}(f(x)) < \rho$, so $M(f(x))$ can only satisfy less than a ρ fraction clauses of $f(x)$. □

0.3 Equivalence

It turns out that both views are equivalent to a more general idea—the hardness of constraint satisfaction problems.

Definition. An instance ψ of a q`CSP` is a collection of functions $\psi_1, \ldots, \psi_m$ (called constraints) from $\{0,1\}^n$ to $\{0,1\}$ where each ψ_i only depends on q bits. An assignment u of n boolean variables is said to satisfy a constraint ψ_i if $\psi_i(u) = 1$. The value $\mathtt{val}(\psi)$ is the maximum fraction of constraints satisfied by any variable assignment. Here, m is called the size of ψ.

It is clear that 3`Sat` is a subset of 3`CSP`.

Definition. For $q \in \mathbb{N}$, $\rho \leq 1$, define ρ-`GAP`q`CSP` to be the problem: given a q`CSP` instance ψ, decide whether $\mathtt{val}(\psi) = 1$ (yes instance) or $\mathtt{val}(\psi) < \rho$ (no instance). We do not care about the cases where the value is in between ρ and 1, hence the "gap".

Theorem (Hardness of `CSP`)**.** *There exists a constant* $q \in \mathbb{N}$ *and* $\rho < 1$ *such that* ρ-`GAP`q`CSP` *is* $\mathbf{NP}$-hard.

Here, $\mathbf{NP}$-hard just means for any $\mathbf{NP}$ language, there is a polynomial time function that takes binary strings into instances of ρ-`GAP`q`CSP` such that $f(x)$ is a yes instance of ρ-`GAP`q`CSP` if x is in L; otherwise $f(x)$ is a no instance.

We will show why all three theorems are equivalent.

Theorem. *The probabalistically checkable proof view is equivalent to the hardness of* `CSP` *view.*

Proof. ($\longrightarrow$). Suppose $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$. We show that $1/2$-`GAP`q`CSP` is $\mathbf{NP}$-hard for some q. Let $L \in \mathbf{NP}$. By our assumption, there is a verifier V for L that uses $c \log n$ random bits and queries the proof q times. Let $x \in \{0,1\}^*$ and $r \in \{0,1\}^{c \log n}$. Let $V_{x,r}(\pi)$ be 1 if and only only if V outputs 1 on x, π with random bits r. Notice that $V_{x,r}$ is a boolean function on qn^c variables but it only depends on q of them. Let $\psi = \{V_{x,r}\}_{r \in \{0,1\}^{c \log n}}$. Then, we see that $\mathtt{val}(\psi) = 1$ if $x \in L$ (since there is a proof π that makes V accept no matter what r is); and $\mathtt{val}(\psi) \leq 1/2$ if $x \notin L$ (since for all proofs π, the probability of V accepting is at most $1/2$).

($\longleftarrow$). Suppose there is some q, ρ such that ρ-`GAP`q`CSP` is $\mathbf{NP}$-hard. Then, given $L \in \mathbf{NP}$, a $\mathbf{PCP}$ system for L works as follows. Let f be the poly-time reduction from L to ρ-`GAP`q`CSP`. On x, the verifier expects a proof π to be variable assignments of $f(x) = \{\psi_i\}_{i=1}^m$. It randomly generates a number $1 \leq i \leq m$ and makes q queries to π to check ψ_i is satisfied. If $x \in L$, then $\mathtt{val}(f(x)) = 1$, so ψ_i is always satisfied for some assignment π. If $x \notin L$, then $\mathtt{val}(f(x)) \leq \rho$, so ψ_i is satisfied with probability at most ρ for any π. By repeating this, the rejecting probability can be reduced to $1/2$. □

Theorem. *The hardness of approximation view is equivalent to the hardness of* `CSP` *view.*

Proof. ($\longrightarrow$) Since 3`Sat` is a special case of `CSP`, this direction is automatic.

($\longleftarrow$) Let $\epsilon > 0$ and $q \in \mathbb{N}$ be such that $(1-\epsilon)$-`GAP`q`CSP` is $\mathbf{NP}$-hard. Let ψ be an instance of $(1-\epsilon)$-`GAP`q`CSP` with n variables and m constraints. Notice that each constraint in ψ can be written as a boolean formula that is the conjunction of 2^q clauses, with each clause containing at most q literals. Doing this for every constraint, we get a formula ψ'. If ψ is a yes instance, there is an assignment satisfying all the clauses of ψ'; if ψ is a no instance, there is at least $\epsilon/2^q$ fraction of clauses not satisfied for every assignment. Recall that each clause of q literals can be transformed into q clauses with each clause being at most size 3 (this was in the proof of `Sat` $\leq_m$ 3`Sat`). Do this for every clause in ψ', we get ψ''. If ψ is a yes instance, then ψ'' is a satisfiable 3`Sat` instance; if not, ψ'' satisfies at most $1 - \epsilon/(q2^q)$ fraction of clauses. This completes the proof. □

0.4 A Proof of the Baby PCP Theorem

We will show the following baby version of the **PCP** theorem.

Theorem (Baby **PCP**)**.** **NP** $\subseteq$ **PCP**$(poly(n), 1)$.

To show this theorem, notice that it is enough we show an **NP**-complete language is in **PCP**$(poly(n), 1)$. We will do this for the language `QuadEq`: the language of satisfiable systems of quadratic equations in $\mathbb{F}_2$. For example,

$$u_1u_2 + u_3u_4 = 1$$
$$u_3u_1 = 0$$

is satisfiable by setting $u_1 = 1, u_2 = 1, u_3 = 0, u_4 = 0$. Its **NP**-completeness can be seen by reducing circuit satisfiability to it. Each wire in the circuit would correspond to a variable, and $+, \cdot$ can clearly be converted to arithmetic in $\mathbb{F}_2$.

Notice that in $(\mathbb{F}_2)^n$, $0^2 = 0$ and $1^2 = 1$. So we can assume each term in the quadratic equations is of the form u_iu_j with $i \neq j$. So a system of m equations over n variables can be thought of as a $m \times n^2$ boolean matrix A specifying the coefficients of u_iu_j and $b \in (\mathbb{F}_2)^m$ specifying the right-hand sides.

A tool we will need is the *Walsh-Hadamard* code—a way to encode binary strings as boolean function. More specifically, given $u \in (\mathbb{F}_2)^n$, its encoding $wh(u)$ is the truth table of $x \mapsto x \cdot u$, where $\cdot$ is the dot product mod 2. This truth table has length 2^n. A key fact about them is the *random subsum principle*: if $u \neq v$ in $(\mathbb{F}_2)^n$, then for half of the vectors x in $(\mathbb{F}_2)^n$, $u \cdot x \neq v \cdot x$. This follows from basic linear algebra. Observe that $u \cdot x = v \cdot x$ if and only if $(u - v) \cdot x = 0$ if and only if $x \in (u - v)^\perp$, which is a $n - 1$ dimensional subspace of $(\mathbb{F}_2)^n$.

This random subsum principle says something crucial about Walsh-Hadamard codes: if two vectors are different, then their code differs in half the bits!

Definition. Given $\rho \in [0, 1]$ and functions $f, g : (\mathbb{F}_2)^n \to \mathbb{F}_2$, we say they are ρ-close if at least they agree on a fraction of at least ρ vectors.

Given a vector in $(\mathbb{F}_2)^{2^n}$, we want to test if it encodes a linear function (i.e if it is of the form $wh(u)$ for some $u \in (\mathbb{F}_2)^n$). It turns out we can do this right most of the time.

Theorem (Linearity Testing)**.** *Let $f : (\mathbb{F}_2)^n \to \mathbb{F}_2$ be such that*

$$\Pr_{x,y \in (\mathbb{F}_2)^n}[f(x + y) = f(x) + f(y)] \geq \rho$$

for some $\rho > 1/2$. Then f is ρ-close to a linear function.

The contrapositive of this statement is useful for us. If f is not ρ-close to any linear function, then the probability the above equality holds for random x and y is less than ρ. We can sample random vectors x_i, y_i many (but constant) times and with high probability one pair does not satisfy the equality. So we can reject with good probability (say 1/2) a function that is not ρ-close to a linear function.

Suppose f is ρ-close to a linear function, what condition does ρ need to satisfy for this function to be unique? The answer is $\rho > 3/4$: say f is $> 3/4$-close to linear functions g_1 and g_2; then, among more than 3/4 of the values f and g_1 agree on, less than 1/4 of them g_2 can disagree with; so g_1 and g_2 must agree on more than 1/2 of values, which means $g_1 = g_2$ (random subsum principle!).

Let $\delta < 1/4$. Given a potentially corrupted code (meaning that it is not linear) f that is $(1 - \delta)$-close to some linear $\hat{f}$, we would like to recover $\hat{f}$. This can be done with high probability as follows. Given x, from which we wish to compute $\hat{f}(x)$, randomly generate x' and set $x'' = x - x'$; output $f(x') + f(x'')$. By union bound, the output is $\hat{f}(x)$ with probability at least $1 - 2\delta$.

With the above machinery, we are ready to prove the theorem.

Proof of Baby **PCP**. Let $A \in (\mathbb{F}_2)^{m \times n^2}$ and $b \in (\mathbb{F}_2)^m$ be an instance of `QuadEq`. The verifier expects the proof to be the Walsh-Hadamard code f of a vector $u \in (\mathbb{F}_2)^n$ and g of its tensor product $u \otimes u$. This means $f \in \{0, 1\}^{2^n}$ and $g \in \{0, 1\}^{2^{n^2}}$. We can identify $u \otimes u$ with the vector $(u_iu_j)_{i,j} \in (\mathbb{F}_2)^{n^2}$. The verifier expects u to contain all the variable assignments, hence $u \otimes u$ should be a vector that contains all the different values of quadratic monomials in the system. This means $A(u \otimes u)$ should be b for a satisfying assignment.

Of course, none of the above can be checked deterministically since that would take too many queries to the proof! Instead, we do the following.

1. Check if f and g are 0.999-close to linear. If not, reject. By our comment on linearity testing, this can be done while only accessing the proof a constant number of times such that if f or g are not 0.999-close, we reject with probability at least 1/2. This means we already met the goal if the proofs are not 0.999-close to linear (to reject $\geq 1/2$ of the time). So in the following we can assume f is 0.999-close to a linear function $\hat{f} = x \mapsto x \cdot u$ and g to $\hat{g} = x \mapsto x \cdot v$.

   Also, since f and g are 0.999-close to linear, if in the following steps we "query" (quoted since we can only do so with high probability as discussed before the proof) bits of $\hat{f}$ and $\hat{g}$ at most 20 times, then with probability $1 - 2(0.001)(50) = 0.9$ the queries came out correct. If in the next few steps we are able to reject a false proof with probability at least t, then in fact we would have rejected it with with probability at least $0.9t$, conditioning on the fact that the queries came out correct.

2. Check if $v = u \otimes u$. To do this, we randomly take $r, r' \in (\mathbb{F}_2)^n$ and check if $\hat{f}(r)\hat{f}(r') = \hat{g}(r \otimes r')$. Let $U = (u_i u_j)_{i,j}$ be $u \otimes u$ written in matrix form, and $V = (v_{i,j})$ be v in matrix form. Then, we see that $\hat{f}(r)\hat{f}(r') = rUr'$ and $\hat{g}(r \otimes r') = rVr'$. Note that if the ith columns of U and V are different, then the ith entry of rW and rV are different for half the r's. So for at least half the r's, $rW \neq rV$. Fix such an r, the probability of r' satisfying $rUr' \neq rVr'$ is at least 1/2. Thus, at least 1/4 of all pairs of r, r' satisfy $rUr' \neq rVr'$ if $U \neq V$. Repeat this 8 times, the probability that $U = V$ but we missed it is $(3/4)^8 \simeq 0.1$. So with probability 0.9 we catch it. Notice that we queried $\hat{f}$ and $\hat{g}$ $3(8) = 24$ times.

3. Check if $Av = b$. Notice that computing Av would require calling $\hat{g}$ m times, which is bad. Instead, we randomly take $r \in (\mathbb{F}_2)^m$ and check $(rA)v = rb$. By the subsum principle, if $Av \neq b$, $(rA)v \neq rb$ half the time. Do this for 10 times so that we catch it with probability at least 0.9.

If A, b is satisfiable, then none of the above would reject the correct proof. On the other hand, if A, b is not satisfiable, then that means no u exists such that $A(u \otimes u) = b$ and any proof will fail with probability at least $0.9(0.9) \simeq 0.8$. This proves $\mathbf{NP} \subseteq \mathbf{PCP}(poly(n), 1)$. □